

Exercise work consists of three parts. After each part, take the mini-exam available in Moodle. The questions concern the exercise so read the instructions carefully and make sure you remember to cover all given tasks.

After the deadline of each mini-exam, example results will be published. This helps you to move to the next part. However, take into account that the deadlines are thus strict, no extensions will be granted!

Two exam points can be acquired from each mini-exam. All three parts of the exercise are compulsory. Completed exercise work from previous years does not apply for this course.

Deadlines for the mini-exams:

- Part 1: Fri 9.2.2024 at 23:59
- Part 2: Fri 23.2.2024 at 23:59
- Part 3: Fri 8.3.2024 at 23:59

If you encounter problems, Google first and if you can't find an answer, ask for help

- Moodle area for questions
- pekavir@utu.fi (<mailto:pekavir@utu.fi>)

Grading

The exercise covers a part of the grading in this course. The course exam has 5 questions, 6 points of each. Exercise gives at maximum 6 points, i.e. the total score is 36 points.

1 Part 1

Read the original research article:

İ. Çınar and M. Koklu. Identification of rice varieties using machine learning algorithms. Journal of Agricultural Sciences, 28(2):307–325, 2022. doi: 10.15832/ankutbd.862482.

<https://dergipark.org.tr/en/download/article-file/1513632>
(<https://dergipark.org.tr/en/download/article-file/1513632>)

There will be questions regarding the article in the mini-exam.

Mini-exam questions:

1. The camera used to take the rice images was placed in a closed box. Why?

Answer: To ensure the same lighting environment for all rice samples.

2. Which morphological feature is: The number of pixels in the smallest convex polygon that can accommodate the rice grain area?

Answer: Convex_Area

3. How many color spaces were used to derive the color features in the original article?

Answer: 5

1.1 Introduction

Will be written in Part 3.

1.2 Preparations of the data

Make three folders in your working folder: 'notebooks', 'data' and 'training_data'. Save this notebook in 'notebooks' folder.

Import all the packages needed for this notebook in one cell (add the ones you apply):

```
In [26]: 1 import numpy as np
          2 import pandas as pd
          3 from matplotlib import pyplot as plt
          4 import cv2 as cv
          5
          6 import glob, os
          7 import itertools
          8
          9 from random import sample
         10 from random import seed
         11
         12 from scipy.stats import skew
         13 from scipy.stats import kurtosis
         14
         15 from IPython.display import display, HTML
         16 display(HTML("<style>.container { width:100% !important; }</style>"))
```

Import the images. Data can be found from (downloading starts as you press the link)

<https://www.muratkoklu.com/datasets/vtdhnd09.php>

(<https://www.muratkoklu.com/datasets/vtdhnd09.php>)

Save the data folders 'Arborio', 'Basmati', 'Ipsala', 'Jasmine', and 'Karacadag' in 'data' folder. Take a random sample of 100 images from each rice species (i.e. 500 images in total). Use seed(50) for enabling reproducible results.

```

In [2]: ▶ 1 # import data
          2 # set the seed for enabling
          3 # the reproduction with the same sequence
          4 seed(50)
          5
          6 path = '../data'
          7 folders = ['Arborio',
          8             'Basmati',
          9             'Ipsala',
10             'Jasmine',
11             'Karacadag']
          12
          13 subset = []
          14 for folder in folders:
          15     path_folder = os.path.join(path, folder)
          16     # all .jpg files from the given folder
          17     files = glob.glob(os.path.join(path_folder, '*.jpg'))
          18     # gather all sampled filenames in subset list
          19     subset.extend(sample(files, 100))

```

Gather the sampled images in a list.

```

In [3]: ▶ 1 # gather images
          2 ims = []
          3 for filename in subset:
          4     # image in BGR order
          5     im = cv.imread(filename)
          6     ims.append(im)

```

We will test the needed functions with one test image. It is 'basmati (244).jpg'. Save this image as test_image.

```

In [4]: ▶ 1 test_image_file_name = '../data/Basmati/basmati (244).jpg'
          2 test_image = cv.imread(test_image_file_name)

```

Determine the contour of each rice (use *findContours* from OpenCV).

Determine the contour also for the test_image.

Plot the original image of the test_image and also its image including the contour.

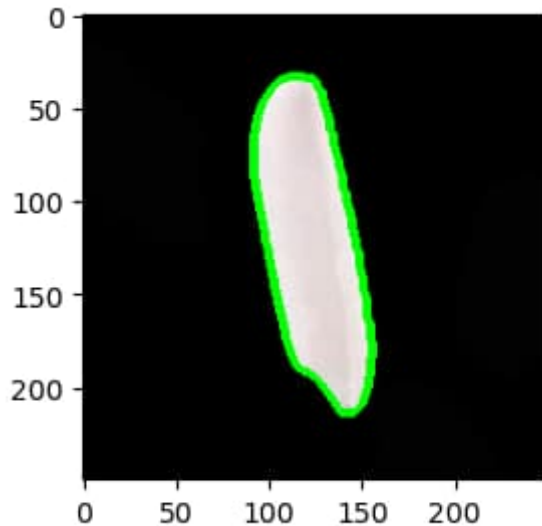
Be aware that *drawContours* adds the contour to the original image, so use the copy of the test_image as input for the function!

```

In [5]: ▶ 1 # https://docs.opencv.org/4.x/d4/d73/tutorial\_py\_contours\_begin.html
          2 def find_contour(im):
          3     imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
          4     ret, thresh = cv.threshold(imgray, 127, 255, 0)
          5     contours, hierarchy = cv.findContours(thresh,
          6                                         cv.RETR_TREE,
          7                                         cv.CHAIN_APPROX_SIMPLE)
          8     cnt = contours[0]
          9
10     return(cnt.reshape(-1,2))

```

```
In [6]: ▶ 1 # find contour for the test_image
2 test_image_cnt = find_contour(test_image)
3 plt.figure(figsize=(3,3))
4 # drawContours arguments:
5 #image, contours, index of contours, color, thickness
6 plt.imshow(cv.drawContours(test_image.copy(),
7                             [test_image_cnt], 0, (0,255,0), 3));
```

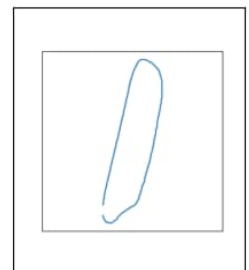
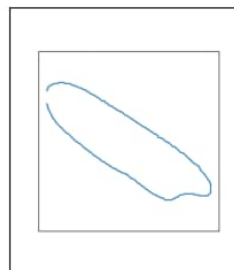
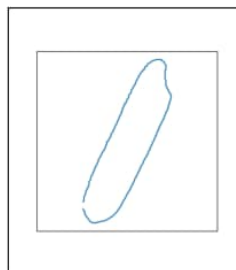
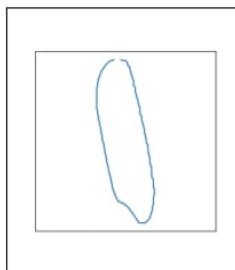


Mini-exam question:

4. Compare the contours below. Which one is the contour of the test_image?

Answer: See this image below, the contour is the first one (you may have had the images in different order in your mini-exam).

```
In [29]: ▶ 1 plt.figure(figsize=(12,3))
2 for i, im_opt in enumerate(['opt1.png',
3                             'opt2.png',
4                             'opt3.png',
5                             'opt4.png']):
6     plt.subplot(1,4,i+1)
7     im_opt = plt.imread('opt{}.png'.format(i+1))
8     plt.imshow(im_opt)
9     plt.axis('equal')
10    plt.xticks([])
11    plt.yticks([])
```



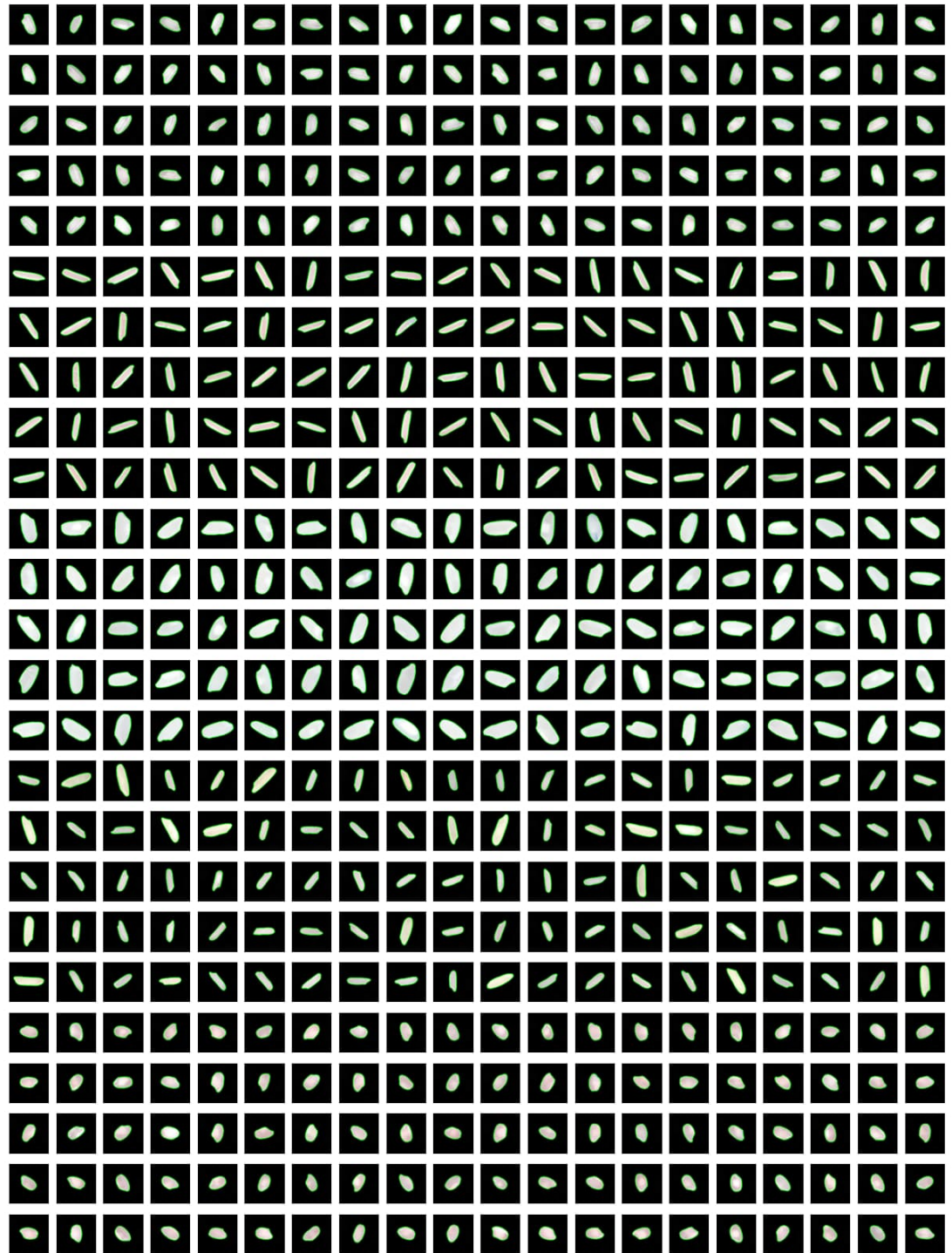
In [8]: ▶

```
1 # find contour for each image in the subset
2 contour_images = []
3 cnts = []
4 ims_with_contours = []
5 for im in ims:
6     # take a copy of the original image
7     # in order to avoid adding the contour to the original image
8     im = im.copy()
9     cnt = find_contour(im)
10    cnts.append(cnt)
11    ims_with_contours.append(cv.drawContours(im,
12                                           [cnt],
13                                           0, (0,255,0), 3))
```

```

In [9]: ▶ 1 # just to check that all of the contours look reasonable
          2 i = 0
          3 fig = plt.figure(figsize=(12, 16))
          4 for im in ims_with_contours:
          5     i+=1
          6     plt.subplot(25, 20, i)
          7     plt.imshow(im)
          8     #get current axes
          9     ax = plt.gca()
         10     #hide axes
         11     ax.get_xaxis().set_visible(False)
         12     ax.get_yaxis().set_visible(False)
         13 fig.tight_layout(pad=0.4);

```



1.3 Feature extraction

Gather the color feature data (12 features):

First, convert each RGB image to YCbCr-image. You can use *cvtColor* function from OpenCV with appropriate color conversion.

Be aware that openCV default channel order is BGR not RGB. In which order does the *cvtColor* function output the components?

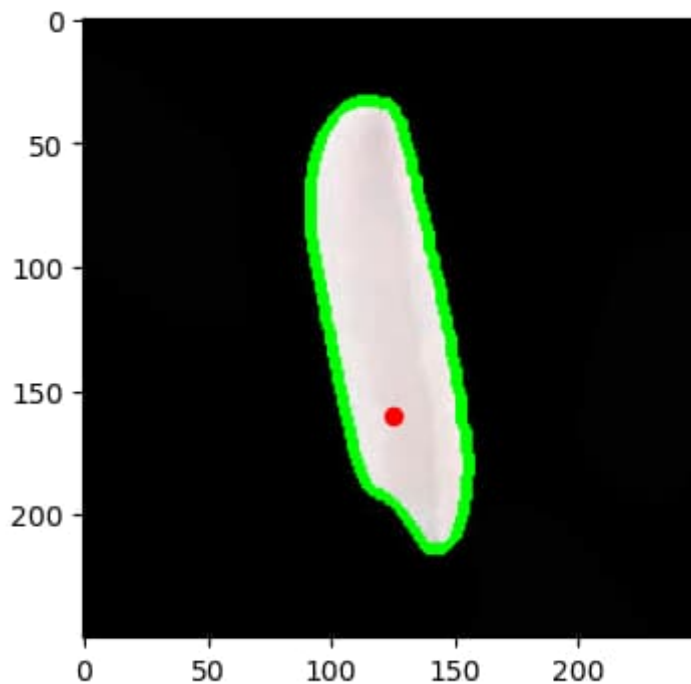
Calculate the following color features for each image, including only the pixels within the contour (you can use *pointPolygonTest* from OpenCV)

- Mean for each YCbCr component
- Variance for each YCbCr component
- Skewness for each YCbCr component
- Kurtosis for each YCbCr component

Is point $x = 125$, $y = 160$ within the contour in the *test_image*? (x is in horizontal direction, y is in vertical direction)

```
In [10]: 1 x = 125
          2 y = 160
          3
          4 print(cv.pointPolygonTest(test_image_cnt, (x,y), False))
          5 plt.figure(figsize=(4,4))
          6 plt.imshow(cv.drawContours(test_image.copy(),
          7                               [test_image_cnt], 0, (0,255,0), 3))
          8 plt.plot(x,y, 'ro');
```

1.0



Mini-exam question:

5. Is the point $x = 125$, $y = 160$ within the contour of the test_image? (x = horizontal pixel, y = vertical pixel)

Answer: Yes

```
In [11]: ▶ 1 def calc_color_features(im, cnt, component):
2
3     # change from BGR into YCrCb color space
4     im_ycrCb = cv.cvtColor(im, cv.COLOR_BGR2YCrCb)
5
6     xs = range(0, im.shape[1])
7     ys = range(0, im.shape[0])
8
9     xys = list(itertools.product(xs, ys))
10
11    # gather the pixels which are within the contour to a list
12    comp_values = []
13    for x, y in xys:
14
15        # check if the point (x,y) is within the contour.
16        # Pixels in image [y, x, color_channel]
17        if cv.pointPolygonTest(cnt, (x,y), False) == 1:
18            comp_values.append(im_ycrCb[y, x, component])
19
20    return(np.mean(comp_values),
21           np.var(comp_values),
22           skew(comp_values),
23           kurtosis(comp_values))
```

What are the mean values of Y, Cb and Cr components for the test image (within the contour)?

```
In [12]: ▶ 1 print('Y:', calc_color_features(test_image, test_image_cnt, 0)[0])
2 print('Cb:', calc_color_features(test_image, test_image_cnt, 2)[0])
3 print('Cr:', calc_color_features(test_image, test_image_cnt, 1)[0])
```

Y: 223.9083980044346
Cb: 133.6614467849224
Cr: 126.87860310421286

Miniexam question:

6. The mean value of Cr component is...

Answer: Almost the same as Cb component value.

Gather the dimension feature data (8 features)

- Fit an ellipse to the contour points (use *fitEllipse* from OpenCV)
- Plot one example image of each rice species including the fitted ellipse
- Calculate the following features for each image (for details, see the original article)
 - the major axis length the ellipse
 - the minor axis length of the ellipse
 - area inside the contour (use *contourArea* from OpenCV)
 - perimeter of the contour (use *arcLength* from OpenCV)
 - equivalent diameter
 - compactness

- shape factor 1 (SF1)
- shape factor 2 (SF2)

Calculate the dimension feature values for the test_image as well. What are the dimension feature values for the test_image?

```
In [13]: 1 def calc_dimension_features(im, cnt):
2
3     # fit an ellipse to the contour.
4     # Ellipse is a tuple including
5     # (center coordinates, axes lengths, angle)
6     ellipse = cv.fitEllipse(cnt)
7     (cc, (a1, a2), ang) = ellipse
8
9     major_a = max(a1, a2)
10    minor_a = min(a1, a2)
11
12    fig = cv.ellipse(im.copy(), ellipse, (0,0,255), 3)
13
14    area = cv.contourArea(cnt)
15    perimeter = cv.arcLength(cnt, True)
16
17    equivalent_diameter = np.sqrt((4 * area) / (np.pi))
18    compactness = equivalent_diameter / major_a
19
20    shape_factor1 = major_a / area
21    shape_factor2 = minor_a / area
22
23
24    return(pd.DataFrame(
25        data={'major_axis_length': [major_a],
26              'minor_axis_length': [minor_a],
27              'area': [area],
28              'perimeter': [perimeter],
29              'equivalent_diameter': [equivalent_diameter],
30              'compactness': [compactness],
31              'shape_factor1': [shape_factor1],
32              'shape_factor2': [shape_factor2]}), fig)
```

```
In [14]: 1 calc_dimension_features(test_image, test_image_cnt)[0]
```

Out[14]:

	major_axis_length	minor_axis_length	area	perimeter	equivalent_diameter	compactness
0	194.516296	50.020149	7409.5	430.835568	97.129133	0.4

Gather all the features in a dataframe: one sample in one row, including all feature values in columns.

For each data point, include also information of the original image and the label (rice species).

Save the data in training_data folder as a parquet file.

```
In [15]: ▶ 1 def gather_color_features(im, cnt):
2
3     df = []
4
5     for component, abbr in enumerate(['y', 'cr', 'cb']):
6
7         c_mean, c_var, c_skew, c_kurt = calc_color_features(
8             im, cnt, component)
9
10        df.append(pd.DataFrame(
11            data={'mean_{}'.format(abbr) : [c_mean],
12                'var_{}'.format(abbr) : [c_var],
13                'skew_{}'.format(abbr) : [c_skew],
14                'kurt_{}'.format(abbr) : [c_kurt]
15            }))
16
17    df = pd.concat(df, axis=1)
18
19    return(df)
```

```

In [16]: ▶ 1 # calculate all features and gather them as a dataframe
2 df = []
3 ims_with_ellipses = []
4 for cnt, filename, im in zip(cnts, subset, ims):
5     # color features
6     df1 = gather_color_features(im, cnt)
7
8     # dimension features
9     df2, fig = calc_dimension_features(im, cnt)
10    ims_with_ellipses.append(fig)
11
12    df3 = pd.concat([df1, df2], axis=1)
13    df3['class'] = [filename[8:11]]
14
15    df.append(df3)
16    print('Features derived for', filename)
17
18 df = pd.concat(df, axis=0, ignore_index=True)
19 df['class_int'] = df['class'].map(
20     {'Arb': 0, 'Bas':1, 'Ips':2, 'Jas': 3, 'Kar':4})

```

```

Features derived for ../data\Arborio\Arborio (3834).jpg
Features derived for ../data\Arborio\Arborio (9115).jpg
Features derived for ../data\Arborio\Arborio (13922).jpg
Features derived for ../data\Arborio\Arborio (1867).jpg
Features derived for ../data\Arborio\Arborio (5898).jpg
Features derived for ../data\Arborio\Arborio (9634).jpg
Features derived for ../data\Arborio\Arborio (13573).jpg
Features derived for ../data\Arborio\Arborio (671).jpg
Features derived for ../data\Arborio\Arborio (3475).jpg
Features derived for ../data\Arborio\Arborio (7806).jpg
Features derived for ../data\Arborio\Arborio (14865).jpg
Features derived for ../data\Arborio\Arborio (11253).jpg
Features derived for ../data\Arborio\Arborio (4438).jpg
Features derived for ../data\Arborio\Arborio (14678).jpg
Features derived for ../data\Arborio\Arborio (13306).jpg
Features derived for ../data\Arborio\Arborio (6472).jpg
Features derived for ../data\Arborio\Arborio (4706).jpg
Features derived for ../data\Arborio\Arborio (11254).jpg
Features derived for ../data\Arborio\Arborio (12258).jpg

```

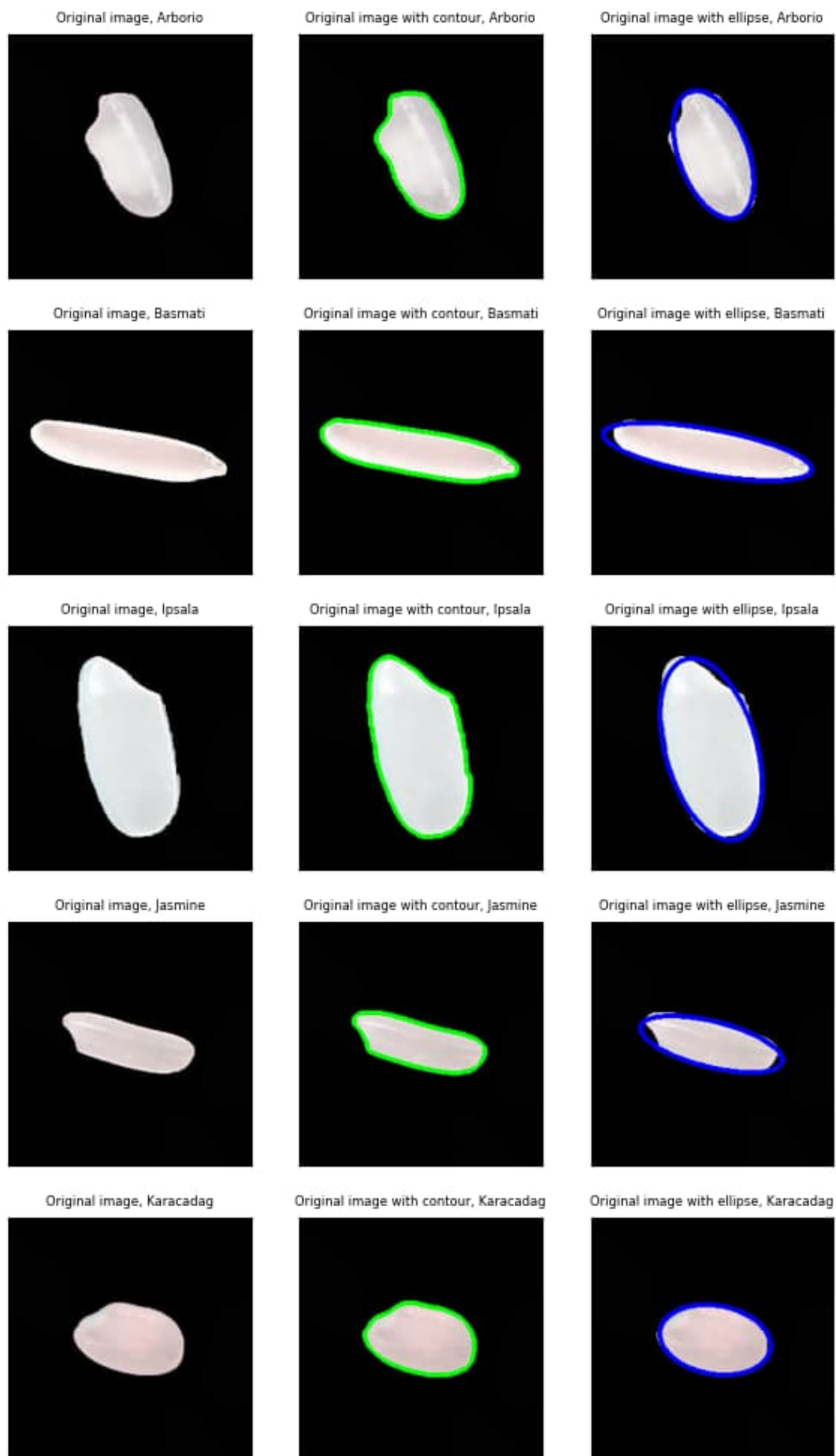
```

In [17]: ▶ 1 df.to_parquet('../training_data/rice_feature_data.parquet')

```

In [18]: ▶

```
1 # plot one example of an original image and
2 # the original image with the contour and
3 # the original image with the ellipse
4 plt.figure(figsize=(6, 10))
5 chosen_images = [0, 100, 200, 300, 400]
6 roll = 0
7 for i, species in zip(chosen_images, folders):
8     roll+=1
9     im = ims[i]
10    imc = ims_with_contours[i]
11    ime = ims_with_ellipses[i]
12
13    plt.subplot(5,3,roll)
14    plt.title('Original image, {}'.format(species), fontsize=6)
15    plt.imshow(im)
16    plt.xticks([])
17    plt.yticks([])
18
19    roll+=1
20
21    plt.subplot(5,3,roll)
22    plt.title('Original image with contour, {}'.format(species),
23             fontsize=6)
24    plt.imshow(imc)
25    plt.xticks([])
26    plt.yticks([])
27
28    roll+=1
29
30    plt.subplot(5,3,roll)
31    plt.title('Original image with ellipse, {}'.format(species),
32             fontsize=6)
33    plt.imshow(ime)
34    plt.xticks([])
35    plt.yticks([])
36
37 plt.tight_layout();
```



Determine the maximum variance of the Cr component for each rice species. Which rice species has the smallest value?

```
In [19]: 1 df.groupby('class').var_cr.max()
```

```
Out[19]: class
Arb      1.599300
Bas      7.692872
Ips      5.612974
Jas     15.464937
Kar      1.858648
Name: var_cr, dtype: float64
```

Mini-exam question:

7. Determine the maximum variance of the Cr component for each rice species. Which rice species has the smallest value?

Answer: Arborio

Determine the minimum equivalent diameter for each rice species. Which rice species has the largest value?

```
In [20]: 1 df.groupby('class').equivalent_diameter.min()
```

```
Out[20]: class
Arb      81.223708
Bas      81.805544
Ips     113.647446
Jas      78.513625
Kar      81.011809
Name: equivalent_diameter, dtype: float64
```

Mini-exam question:

8. Determine the minimum equivalent diameter for each rice species. Which rice species has the largest value?

Answer: Ipsala

Determine the minimum, maximum and median equivalent diameter for Basmati rice samples.

Compare the equivalent diameter value of the test_image to these values.

```
In [21]: 1 df[df.class_int == 1].equivalent_diameter.describe()
```

```
Out[21]: count    100.000000
mean      95.734143
std       5.637933
min       81.805544
25%      92.061480
50%      96.444960
75%      99.973800
max     106.250546
Name: equivalent_diameter, dtype: float64
```

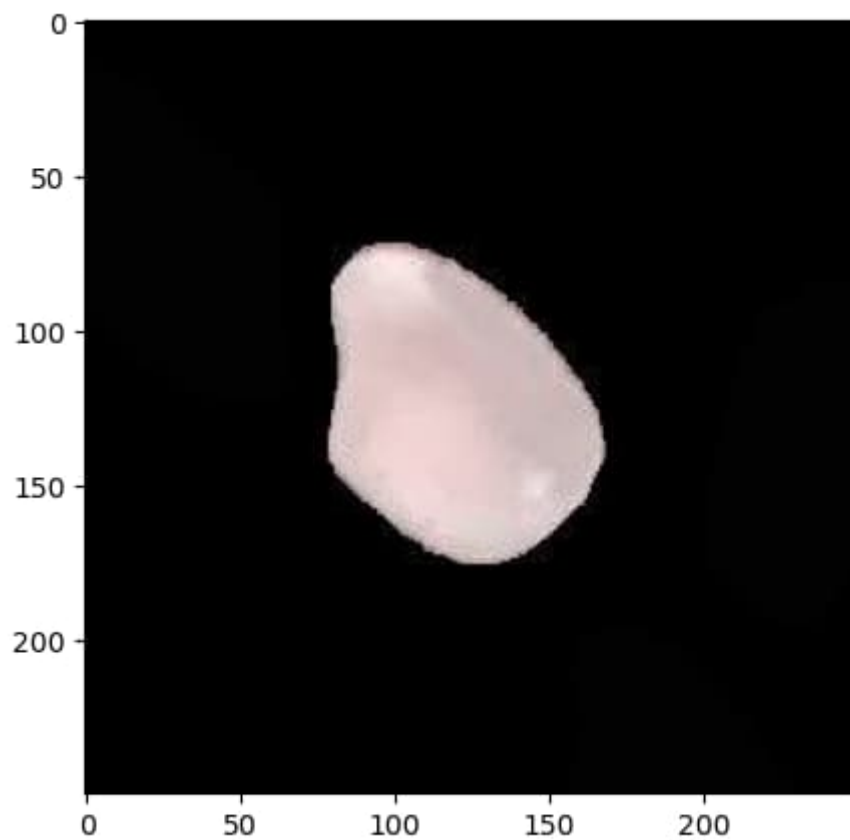
Mini-exam question:

9. Determine the minimum, maximum and median equivalent diameter for Basmati. Where in this range does the equivalent diameter of the test_image lie?

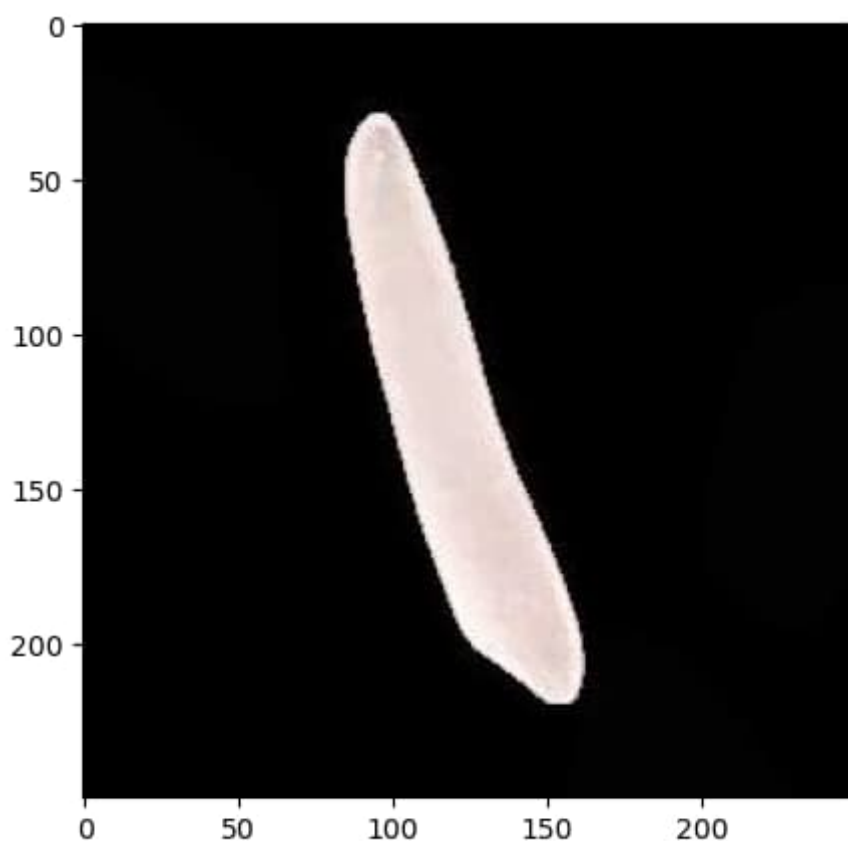
Answer: The equivalent diameter for the test image is 97 pixels. Thus, the value is close to the median value of the equivalent diameter for Basmati samples.

Does a high value of compactness mean that the rice is rather round or thin?

```
In [22]: ▶ 1 # a high value of compactness  
2 plt.figure(figsize=(5,5))  
3 plt.imshow(ims[df[  
4     df.compactness == df.compactness.max()].index[0]]);
```



```
In [23]: ▶ 1 # a small value of compactness  
2 plt.figure(figsize=(5,5))  
3 plt.imshow(ims[df[  
4     df.compactness == df.compactness.min()].index[0]]);
```



Mini-exam question:

10. Does a high value of compactness mean that the rice is rather round or thin?

Answer: Round